5-2014

Sonar-Based Autonomous Navigation and Mapping of Indoor Environments Using Micro-Aerial Vehicles

Mark Williams

Sonar-Based Autonomous Navigation and Mapping of Indoor Environments Using Micro-Aerial Vehicles

by

Mark Williams

A Project Report Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
Master of Science
in
Computer Science

Supervised by

Dr. Zach Butler

Department of Computer Science

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York

May 2014

The project "Sonar-Based Autonomous Navigation and Mapping of Indoor Environments Using Micro-Aerial Vehicles" by Mark Williams has been examined and approved by the following Examination Committee:

Dr. Zach Butler
Associate Professor
Project Committee Chair

Dr. Richard Zanibbi
Associate Professor

Dr. James Heliotis
Professor

Contents

Chapter 1

Abstract, Introduction, Prior Work

1.1 Abstract

The purpose of this project is to add autonomous navigation and mapping to a micro aerial vehicle (MAV) with the aid of additional sonar sensors that are mounted to the MAV's chassis. In this paper, an Extended Kalman Filter Simultaneous Localization and Mapping (EKF-SLAM) system is implemented on a MAV with four sonar sensors. Some simulations are developed to test the various modules and processes created for the project, and the final system is tested on an AR.Drone 2.0 quadcopter with an added sonar sensor package.

1.2 Introduction

This project attempts to implement Extended Kalman Filter (EKF) Simultaneous Localization and Mapping (SLAM) on the Parrot AR.Drone 2.0 quadcopter. Simultaneous Localization and Mapping is a method that combines the mobile vehicle's dead-reckoning information and features detected in the environment at the mobile vehicle's location to build a more accurate estimate of the vehicle's state at the given time. A state configuration includes the vehicle's location estimate, the map of detected features, and the correlation between each sensed feature and the vehicle at the current time. The goal is to use sensed information about the environment to overcome error inherent in the drone's odometry and sensors. This project uses the Parrot AR.Drone 2.0 as a platform. The AR.Drone 2.0 is a small (0.517 meters by 0.517 meters) quadcopter that can perform well in indoor environments.

It has an accessible application programming interface (API), which allows remote-control from a variety of applications and devices. It is user-controllable and provides some internal sensor information to the user.

Figure 1.1: *Parrot AR.Drone 2.0 Quadcopter*

Dead-reckoning of the drone's location is done by integrating the drone's reported velocities over time. This becomes increasingly inaccurate the longer the drone is flown because the sample rate is relatively low and velocities can vary between samples. Dead-reckoning errors are cumulative. Additionally, large or sudden changes in the drone's actual position can reduce accuracy of the reported velocities. The goal of the project is to overcome these odometry errors. A custom-built sensor package containing four sonar range-detecting sensors is mounted on the AR.Drone's chassis. Simultaneous Localization and Mapping uses sensor information to detect features in the surrounding environment. These features can be used to overcome error in the base vehicle's localization. Currently available GPS products cannot be used for localization as they are unreliable indoors and their resolution is too low.

1.3 Prior Work

There has recently been a large amount of research in the micro-aerial vehicle (MAV) field. As MAVs become less and less costly, they become more accessible to researchers and hobbyists. Much of this recent research uses visual information, either from cameras mounted on the quadcopter, cameras external to the quadcopter, or both. Saxena, et al. use single camera sources to identify and navigate indoor environments using the cameras mounted on the Parrot AR.Drone 1.0[1]. Kumar, et al. use a Kinect and additional sensors for 3D exploration using a custom-built MAV [2]. Krajnik et al. provide a detailed analysis of the AR.Drone as a research and education platform in their paper[3], with in depth consideration of both the hardware and software of the drone. A large amount of quadcopter related

research involves the use of external cameras to keep track of the vehicles position. Large camera arrays can be used to have incredibly fine control over a vehicle, but it is unfeasible to expect a high density of HD cameras in every environment that the vehicle can encounter. Since this project does not use external cameras, generic sonar related research is examined. In *Mobile Robot Localization Using Sonar*, Drumheller uses a land-based vehicle with a sonar sensor on a rotating platform that takes 360 degree readings of its environment [4]. He uses the resulting sonar image to match locations in a pre-existing map of the environment for localization with excellent results. Varveropoulos used sonar mounted on a ground-based vehicle to build probabilistic occupancy grid maps for localization[5]. Muller and Burgard use sensor fusion between multiple sonar, airflow sensors, and an IMU for particle filter based localization using a remote controlled blimp [6]. The subject of their paper is similar to the subject of this project, as they are used a relatively unstable airship with a small number of sensors for localization. Tardos and Neira provide the basis for this project in their paper, *Robust Mapping and Localization in Indoor Environments using Sonar Data* [7], which used a land-based vehicle with eight sonar sensors to create multiple independent stochastic local maps for map joining. This project attempts to use their mapping system on a MAV platform with four sonar sensors. This adds difficulty to the project, as the MAV has higher odometry error, and further limiting the number of sonar reduces the available data for feature extraction.

Chapter 2

System Overview

The following system overview details first the hardware platform of the system, followed by the software running on all of the hardware components.

2.1 Hardware Platform

Control of the AR.Drone, as well as the rest of this project, is done in a Linux virtual machine running on a standard laptop computer. Communication with the AR.Drone is done using a Python library called *libardrone* [1]. This library comes with most of the communication functionality required implemented, allowing me to quickly test basic control of the AR.Drone from my laptop computer. The AR.Drone and computer communicate over a standard WiFi connection. Four sonar range sensors were mounted on the AR.Drone for the purpose of this project. These sonar are mounted on the left, front left, front right, and right sides of the AR.Drone and are used to gather information about the environment the AR.Drone is in. The sensors communicate with the host computer via XBee[?] wireless radio units. The host machine communicates with the receiving XBee via a USB to Serial interface, and the sonar sensors are attached directly to the analog to digital conversion inputs on the transmitting XBee that is mounted on the AR.Drone. Both the sensors and the XBee wireless radios are very low power, allowing them to run off of the AR.Drone's battery without drastically reducing the AR.Drone's flight time. The host computer communicates with the XBee via a Python library called *python-xbee* [2]. The XBee connected to the computer waits for packets of sensor data sent from the XBee mounted on the AR.Drone. The transmitting XBee converts data from the sonar sensors ten times per second and transmits

[1] https://github.com/venthur/python-ardrone
[2] https://code.google.com/p/python-xbee/

the data to the receiving XBee into the computer.

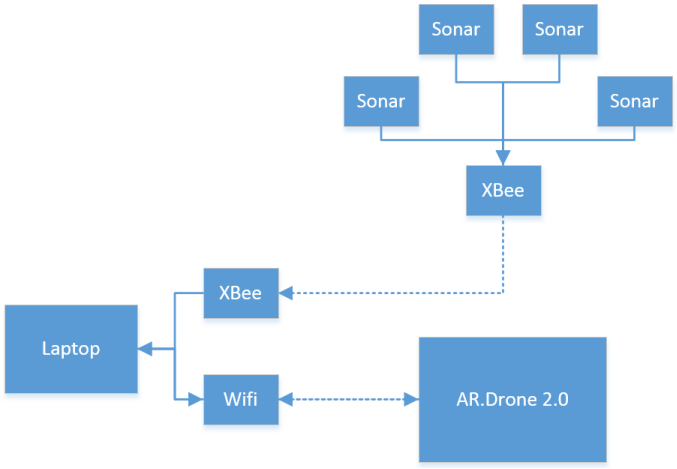

Figure 2.1: *Hardware Platform Block Diagram*

Figure 2.2: *Sensor platform mounted on the AR.Drone 2.0. The XBee control board is circled in blue, and the sonar sensors are circled in green.*

For the purpose of this project, the system considers both the drone and sonar from overhead in a two-dimensional space, ignoring the Z axis. This decreases the complexity of the created maps, and simplifies drone control. Since the drone is primarily moving indoors, it will maintain a height of approximately half a meter off the ground.

2.1.1 Parrot AR.Drone 2.0

The Parrot AR.Drone 2.0 is a durable, robust quadcopter. It comes with forward and bottom facing cameras, bottom facing sonar, a gyroscope, and an accelerometer. The drone hosts a WiFi connection, which a WiFi-enabled device can use to connect to the drone. The drone is 'black-boxed' - the user has access to a number of control commands and limited sensor information. For instance, direct access to the gyroscope or accelerometer is not allowed. Instead, the drone periodically sends a 'navdata' packet, containing information about the drone's velocity, rotation, battery life, and other settings. Parrot has published an application programming interface (API), which allows for simplified control of the AR.Drone. Instead of having to write complex feedback controllers for each of the four rotors, the computer onboard the AR.Drone handles all of the basic control for flight. The API gives access to commands such as forwards, backwards, turn left, and the AR.Drone will translate and execute these simple commands. The AR.Drone API also gives access to information gathered from sensors onboard the AR.Drone. A navdata packet contains information about the AR.Drone's x, y, and z velocities, as well as its current yaw, pitch, and roll. The AR.Drone does not store its own position estimate internally, as the developers decided that simply integrating the velocities over time would be too inaccurate. This is one of the major issues that this project attempts to overcome.

2.1.2 Control computer

All of the command and control code of the system is run in a Ubuntu 12.04 virtual machine on a Lenovo Y500 laptop. The laptop has a Core i7 quad core processor and 8 GB of RAM.

2.1.3 LV-MaxSonar EZ0 Range Finding Sensor

The sonar sensors are used in this project to both navigate and map the environment, which means that both speed and accuracy of readings is important. The sonar sensors used are the LV-MaxSonar EZ0 sonar range finders [8]. These sonar have very low power requirements, drawing only 2 mA at 3.3 volts . The sonar beam has a 30 degree wide cone ($\mathcal{B}$), and can sense ranges from approximately 0.154 meters to 6.5 meters. Objects closer than 0.154 meters return the minimum range reading of 0.154 meters. The sonar sensor outputs updated range readings at 20 Hz. The sensor is quite accurate at range readings when perpendicular to its target, but targets at odd angles or with varied surfaces can cause less accurate readings.

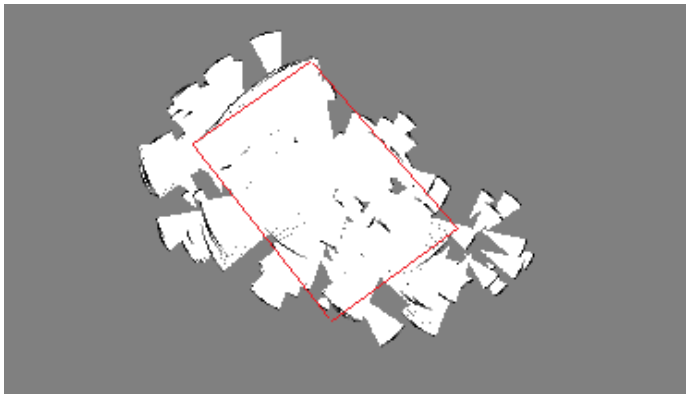

Figure 2.3: *Grid-based probability map of captured sonar data from a flight in a rectangular room. Note the large number of spurious and inconsistent returns. The red box is the approximate size and location of the walls of the room.*

I am using the analog output from the sonar sensors for this project. The analog output produces a voltage between zero volts and the applied input voltage, scaled for the range detected by the sonar. In this system the sonar are powered with 3.3 volts, which means the analog output will read somewhere between 0 and 3.3 volts. However, the XBee Analog to Digital conversion inputs accept a range of 0 to 1.2 volts [**?**], so a simple resistor-based voltage-divider network was created to scale the 3.3 V signal from the sonar to the required 1.2 V input of the XBee. The ADC input on the XBee then converts this to a value from 0 to 1023.

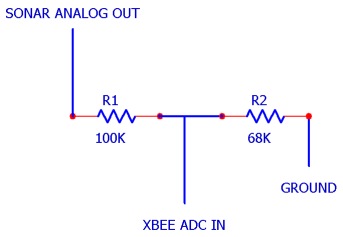

Figure 2.4: *Voltage divider circuit for a single sonar to XBee connection.*

This value is then transmitted to the receiving XBee attached to the host computer, which scales it back to the appropriate value. Finally, the median of every three readings per sensor is taken, to further protect against incorrect sonar readings caused by crosstalk between sonar emissions or objects with irregular surfaces. The following figure depicts the raw range readings from the four sonar sensors mounted to the drone while the drone is off and stationary. The left sensor is approximately 0.5 meters from an obstacle, the front

left and right sensors are clear of obstacles for at least 2 meters, and the right sensor is clear of obstacles for at least 1 meter. In the following graph, you can see that the sonar have a negative skew - they tend to report a shorter range than the actual distance, rather than a longer range.

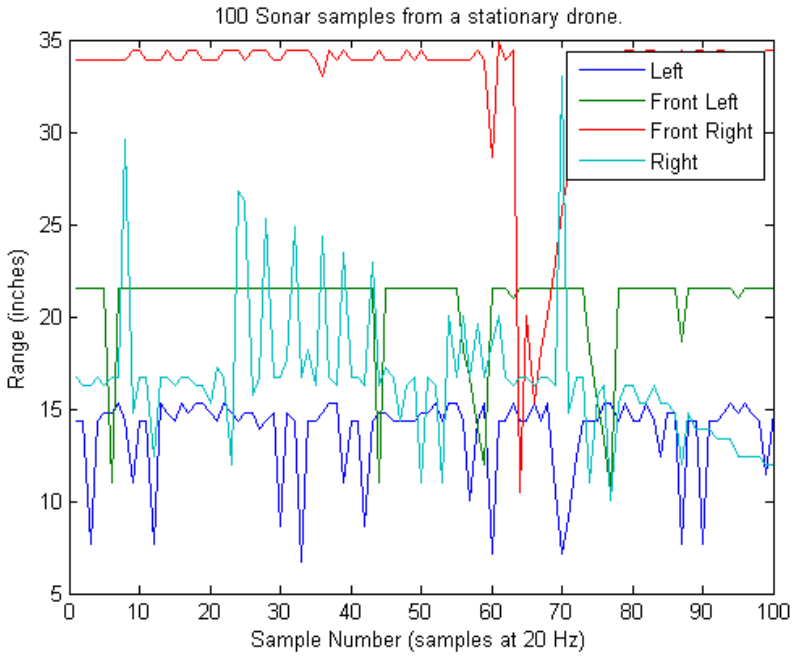

Figure 2.5: *Graph of 100 sonar range samples taken from a stationary drone.*

2.1.4 XBee Wireless Radio

The XBee[?] wireless radios are configured as a transmitter and a receiver. The transmitter is mounted on the AR.Drone, and periodically reads the four sonar sensors connected to its four analog to digital convertor (ADC) inputs. The sensors are connected to the four ADC inputs of the XBee through the previously mentioned resistor-based voltage-divider network that converts the 0 to 3.3 volt signal of the sensor output to the 0 to 1.2 volt that the ADC inputs accept. The transmitting XBee is powered by the AR.Drone's battery, and the receiving XBee is powered by the serial to USB adapter plugged into the control computer. Readings from the sonar sensors are transmitted to the control computer at 6.67 Hz. A Python module that reads serial information from the XBee and extracts the sensor information was created for this project.

2.2 Software Overview

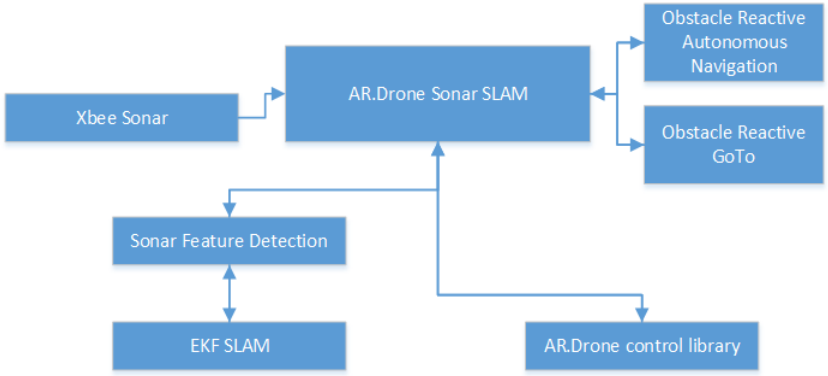

Figure 2.6: *Software Overview Block Diagram*

2.2.1 AR.Drone Sonar SLAM

For this project, a number of software modules were integrated. Some of software was created by a third party and used directly, some was created by a third party and modified, and the some was written specifically for this project. Together these modules communicate with the AR.Drone 2.0, the XBee wireless radios, and the sonar sensors. This communication allows for the drone to be controlled while the system gathers sonar data and extracts feature information for EKF-SLAM. The top level of the project creates and starts the threads for XBee Sonar, Sonar Feature Detection, EKF-SLAM, the AR.Drone 2.0 communication and control library, and the autonomous drone control functions. All software developed for this project is available at `https://bitbucket.org/skudmunky/ardrone-sonar-slam`.

2.2.2 XBee Sonar

A Python module that communicates and translates the XBee serial information was created for this project. The module is responsible for XBee serial communication (done via the *python-XBee* library), and for receiving, unpacking, and converting the sensor readings into usable data. The *python-XBee* library converts incoming data into dictionaries. A packet containing ADC information looks like {["adc-n":1023],...}. The dictionary associates each ADC reading with the appropriate sensor. This module unpacks this dictionary, and scales the ADC data back to the correct range reading with each sensor. Then, the readings are put into a 4 by n array of readings and sorted so a median filter can be applied to the n readings from each sensor.

2.2.3 AR.Drone control library

The AR.Drone API gives access to a number of movement commands. Communication with the drone happens every 50 milliseconds. When the drone is initially launched, it is fully under the operator's control. This enables the operator to ensure that the drone doesn't drift or crash during takeoff, and that it is in a clear area for flight. Once the drone is airborne and stable, the operator can enable the Obstacle Reactive Autonomous Exploration or Obstacle Reactive GoTo programs that have been implemented in this system. The AR.Drone platform makes precise movement control difficult. As the drone is airborne, it can easily slide or drift in any direction. This is quite different from the problems inherent in a wheeled or tracked system, which tends to have error only on the axis of travel. The relative lack of friction in a hovering vehicle versus a ground-based vehicle means that it is difficult to stop instantly, as the drone must decelerate before stopping. It is very difficult for the drone to appropriately stop when an obstacle is detected - it would continue flying forwards due to stored momentum and collide with the obstacle anyway, which would then cause the drone to bounce back and move erratically while attempting to stabilize itself. A number of the AR.Drone 2.0 API commands were used in this project, and are detailed in Appendix B. This project uses the third-party Python library *libardrone* to access the AR.Drone 2.0 API commands. *Libardrone* was extended with a pose estimation module, which integrates the drone's internally reported velocities over time to create an estimate of the drone's position relative to its starting point. This was tested in simulation by giving the pose estimation module x and y velocities at differently timed intervals and viewing the resulting location estimate.

2.2.4 Obstacle Reactive Autonomous Exploration

A state machine was designed for obstacle reactive autonomous exploration (see Appendix A for detail). The state machine attempts to move the drone forwards. If there is an obstacle detected in front of it, the drone will reverse a small amount and rotate 45 degrees towards the clearest side sensor. If the drone is moving forward and a side sensor detects an obstacle, it rotates away from that side sensor and continues moving forwards. In this program, a front obstacle is classified as a range of less than 0.5 meters, and a side obstacle is classified as a range less than 0.4 meters. In the Reverse states, an obstacle is considered 'cleared' if the sensed range is 0.1 meters greater than the appropriate obstacle range.

2.2.5 Obstacle Reactive GoTo

An obstacle-reactive, force-based GoTo was also implemented. This allows testing of the drone's dead reckoning (reported velocities integrated over time) as well as the accuracy of

the EKF-SLAM project as a whole. A desired destination point is given as an X,Y position on the map, and the drone attempts to fly towards this location. First, the drone rotates towards the destination. Then, it flies towards the destination, unless it detects an obstacle. If the obstacle is close enough to interfere with the drone's path, it turns away from the obstacle and continues traveling forward. If the difference between the angle required to reach the destination and the drone's current angle becomes too large, the drone stops moving forward and rotates towards the destination. This process repeats until the drone has reached the desired destination.

Chapter 3

Mapping and Localization

An Extended Kalman Filter Simultaneous Localization and Mapping (EKF-SLAM) system that works with the AR.Drone 2.0 and custom-built sensor platform was designed for this paper. This system is tailored for exploring indoor environments, or other environments where landmark features are simple and relatively close together. This system is based on, but differs slightly, from the system created by Tardos et al. in their work [7]. Their system uses multiple vehicle positions to create a sliding-window of sonar data. This data is used in a Hough transform based feature-extraction system that accumulates votes for two types of features: points and lines. The detected features are then used in the EKF-SLAM update to create a new state configuration of the vehicle.

3.1 Feature Detection

The sonar used in this project are low resolution, due to their large emission cone, and susceptible to crosstalk. Also, moving obstacles such as people can cause unwanted sonar readings. Thus, any feature detection must be able to determine what sonar results are useful, and discard any spurious readings. After some initial tests with the sonar mounted on the drone, it was obvious that grid-based probabilistic maps would not be clear or consistent enough to allow for localization. The sonar results can be inconsistent when the angle to the obstacle is changed, and the low number of sonar present in the system does not allow enough sonar overlap to overcome spurious returns. In order to more accurately detect and classify landmarks using small amounts of sonar, the Hough Transform-based system detailed in Tardos and Neira's work [7] was used. This system has two benefits over a Cartesian system: it is a vote-based system, and it stores landmarks as a tuple of a distance from an origin (ρ), at a specific angle (θ), which reduces the complexity of the

equations used in EKF-SLAM.

3.2 Sonar Modeling

As previously mentioned, the LV-MaxSonar EZ0 has a 30 degree emission cone $\mathcal{B}$. Therefore, the longer the range-reading on the sonar, the larger the uncertainty area for the obstacle. For this project, the system attempts to detect two kinds of geometric landmarks: points and lines. A point could refer to the inside or outside of a corner, and a line would relate to a wall. This set of features is tailored for indoor environments such as hallways and rooms. A smooth wall is detected by the sonar if a perpendicular line from the wall is contained inside the sonar sensor's emission cone. The distance returned by the sonar is the perpendicular distance from the sensor to the line. A wall feature detected by multiple positions will create multiple readings, where the perpendiculars are tangent to the sonar's emission arc. A point feature will produce similar returns for a sonar sensor, except multiple sonar emission arcs will intersect at the location of the point or edge. This project treats the visibility angles for corners, edges, and walls as indistinguishable.

3.3 Hough Transform Feature Detection

The Hough transform is a voting system. Sensors accumulate information about the environment, and store all the associated data in a discrete cell space called Hough space. The cells in Hough space that accumulate the most votes should correspond to the most frequently detected features in the environment. My system keeps track of the sonar associated with each vote, which makes it simple to associate a single feature with groups of sonar returns. Tardos et al. found that associating sonar returns with line and point features is equivalent to finding groups of sonar that are either all tangent to the same line, or that intersect at the same point [7]. A line is stored as the distance from the base origin (ρ) at the angle representing the orientation of the line (θ). A point is similarly stored as the polar representation of its location, as this is compatible with the Hough space.

Algorithm 3.1 *Basic Hough voting algorithm for lines [7].*

```
for i := 1 to num_drone_posititions do
        for j := 1 to num_sensors do
                ;computer sensor location relative to origin
                for alpha in -15 to 15 step 4 do
                        ; computer line parameters and vote
                        theta = sensor angle + alpha
                        rho = range + sensorX*cos(theta)
                                + sensorY*sin(theta)
                        vote(rho, theta, i, j)
                end
        end
end
```

This system operates with a maximum map size of 32 square meters. The origin of map is at (0,0) on the X and Y axis with a rotation of 0. Similarly, the Hough Space has two dimensions: 0 to 32 meters on one axis (ρ), and 0 to 360 degrees for the other axis (θ). For ease of computation and to account for the accuracy of the sonar, drone position, and drone rotation, the vote accumulation bins in Hough Space have been reduced to 160 bins for ρ (0.2 meters per bin) and 90 bins for θ (4 degrees per bin). Each sonar return places votes in the correct bins. The sonar has a $\mathcal{B}$ of 30 degrees, thus the sonar cone extends from $-\mathcal{B}/2$ to $\mathcal{B}/2$. Each sonar produces a constant number of votes, making the Hough process linear with the number of sonar returns. A sensing window is a sliding window of multiple sequential drone positions and their associated sonar returns. This allows for more robust feature detection, as strong environment features are voted for by multiple sensors or positions of sensors.

After the votes have been accumulated, the Hough space is searched for bins that have votes over a certain threshold. The point and line bins that exceed the vote threshold are added to a list and sorted in descending order. If a point and a line are both present at the same bin, the one with a higher number of votes is taken and the other one is ignored. Then, the voted features are matched to the sonar returns that voted for them. Sonar are allowed to vote for a maximum of one landmark, meaning each update window can add a maximum of *window length * number of sensors* landmarks to the global feature map. This was tested in a simulation that combined the *Pose Estimate, Sonar Feature Detection* and *EKF-SLAM* modules created for this system. The simulation depicts a drone moving in a straight line down a hallway that starts at a specified angle. After a number of movements, the angle of the hallway and drone's motion increases by 45 degrees. For the purpose of this test only the side sonars are simulated, to reduce the complexity of the simulation.

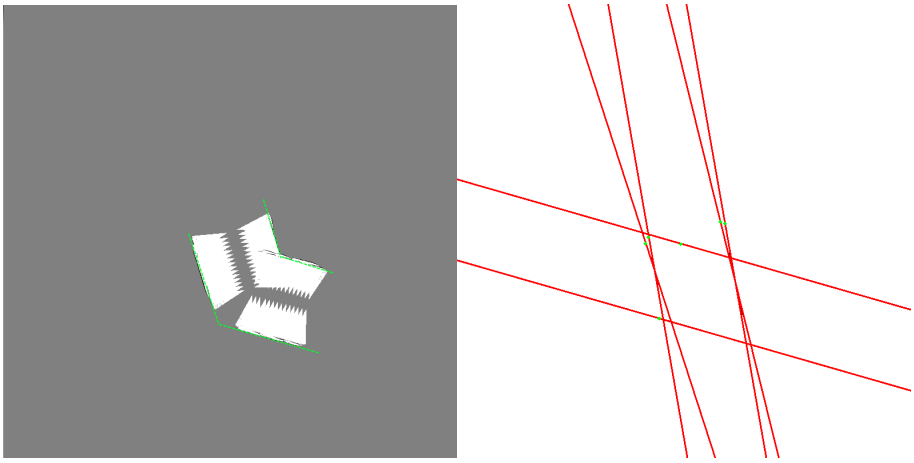

Figure 3.1: *Left: Simulated sonar returns created by drone moving in a straight line through a hallway that has a 45 degree turn (Approximate simulated hallway shown as the dotted green line). Right: The associated detected line features. The red lines are the infinite line voted for by the sensors, and the green dots on the are the location of the (ρ,θ) point in Hough space associated with the line.*

3.4 Map Building and EKF-SLAM

Due to the differences between this platform and the platform used by [7], this implementation of the Extended Kalman Filter (EKF) Simultaneous Localization and Mapping (SLAM) system is slightly changed. The significant changes are explained, for the basis of the system please see Part 3 of [7].

EKF-SLAM attempts to localize a mobile robot in an unknown environment, while building a map of the environment it navigates. The map is a combination of $\hat{\mathbf{x}}^B_{\mathcal{F}_k}$ and $\mathbf{P}^B_{F_k}$, where $\hat{\mathbf{x}}^B_{\mathcal{F}_k}$ is the drone's configuration, containing the estimated drone's location and sensed feature landmarks at step k given the previous step *k-1,* and $\mathbf{P}^B_{F_k}$ is the estimated error covariance of $\hat{\mathbf{x}}^B_{\mathcal{F}_k}$. The basic EKF-SLAM algorithm is:

Algorithm 3.2 *EKF-SLAM Update Equations[7]*

$$
\begin{aligned}
\hat{\mathbf{x}}^B_{\mathcal{F}_k} &= \hat{\mathbf{x}}^B_{\mathcal{F}_{k|k-1}} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}_k(\hat{\mathbf{x}}^B_{\mathcal{F}_{k|k-1}})) \\
\mathbf{P}^B_{\mathcal{F}_k} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}^B_{\mathcal{F}_{k|k-1}} \\
\mathbf{K}_k &= \mathbf{P}^B_{\mathcal{F}_{k|k-1}}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}^B_{\mathcal{F}_{k|k-1}}\mathbf{H}_k^T + \mathbf{R}_k)^{-1}
\end{aligned}
$$

$\mathbf{K}_k$ is the matrix that allows us to do the EKF-SLAM update equations, using the previous state *k-1* and the current step k. $\mathbf{K}_k$ is created by taking the previous covariance $\mathbf{P}^B_{F_{k|k-1}}$, the correlation between sensed landmarks and drone position $\mathbf{H}_k$, and the feature-detection process noise $\mathbf{R}_k$.

EKF-SLAM uses a three step update system. First is the drone's estimated motion - "where do I think I moved?". Second is the feature observation update - "when I moved, what do I think I saw?". Finally, the EKF update equation combines the observed features, estimated movement, and relevant covariance matrices for an updated state estimate.

3.4.1 Map Initialization

At base B of the local map the drone has not moved. The map is initialized with the vehicle's current location relative to the map base, no features in the map, with an error covariance of $\mathbf{0}$.

3.4.2 Robot Motion

As the drone completes an update window, the motion from step *k-1* to step k is estimated by dead reckoning of the drone's odometry with process noise added. The process noise for the drone is assumed independent of the drone's motion, as the flying drone is equally able to move incorrectly or unexpectedly in any direction despite its desired velocities. One important difference between [7] and this approach is that in this approach, the drone's location is already stored relative to the local map. Thus, the 2D transformations detailed in this step of the EKF update by [7] are not required. This reduces the complexity of the calculations required by this step in the update.

3.4.3 Feature Observations

At the end of the update window, each sonar sensor has a range associated with it. The Hough transform associates each sonar with the most appropriate map feature. A new matrix is created, which contains the theoretical distance from each sonar sensor to its associated feature, adding measurement noise to the theoretical sensor readings that is independent of the process noise. This provides the basis for the $\mathbf{R}_k$ and $\mathbf{H}_k$ matrices. $\mathbf{R}_k$ is the measurement noise, which is heuristically set to be a function of the sensed range to the obstacle and the number of votes the obstacle has received. $\mathbf{H}_k$ is the correlation between the drone's position for each sensor and its associated landmark.

After processing a number of steps, the drone's configuration contains an estimate of the sensed map features, their correlation to the drone's position, and an estimate of the drone's location, with odometry errors compensated for by detected map features. It is important to note that multiple tests of the drone in the same area will produce independent maps, as the maps are built relative to the starting drone location.

Chapter 4

Results

4.1 Simulation

The final EKF-SLAM system was tested using a combination of the *Pose Estimation, Sonar Feature Extraction, and EKF-SLAM* modules created for this project. The simulated drone moves at a specified velocity for an amount of time, then increases the angle at which it is traveling by 45 degrees and continues moving at the previous velocity. The sonar are simulated by feeding the *Sonar Feature Extraction* module ranges for each sonar, where the sonar ranges are all a fixed distance, plus or minus a percentage of that distance to simulate noise. In simulation, the EKF-SLAM system performed well, estimating the drone's position within 5% of the position generated by just the odometry. This small difference is due to the size of the discrete bins of the distance and angle of features stored in Hough space. If the drone movement process noise is increased in simulation, the system relies on the feature observation for the EKF update and the difference between the odometry position and the EKF-SLAM result is larger. If the feature detection process noise is increased in simulation, the EKF-SLAM system trusts the drone's odometry more and the difference between the odometry position and the EKF-SLAM result is negligible. This shows that the EKF-SLAM system is working as intended.

4.2 Real-World Experiments

Real-world experiments were done using the Parrot AR.Drone 2.0 with the previously mentioned sonar sensor packaged mounted on the drone's chassis. Experimental flights were done in small room (4 meters by 8 meters), a large room (8 meters by 16 meters), and a hallway (2 meters wide). To perform an experiment, the drone was launched from a location

in the room or hallway and allowed to come to a stable hovering state. The EKF-SLAM system was tested both during autonomous drone flight and while the drone was under user control. After a short flight time (usually under 2 minutes, given the drone's short battery life), the drone was flown back to the origin by the user. This has the added drawback of introducing user error into the system. However, launching the drone is erratic due to the added weight of the sensor package on the drone, so some user input is needed to start and stop the tests. A test starts when mapping is enabled, using the current hovering position of the drone as the base reference $\mathcal{B}$. Upon completion of the test, the approximate difference between the drone's internal position estimate and its location relative to the ground-truth starting point is measured. If the x and y difference between the positions is less than 0.5 meters, the test is considered successful. Unfortunately, real-world tests have shown a flaw in the components chosen for this system. In physical tests with the drone and sensor package, the EKF-SLAM update configuration estimate tends to diverge from the actual ground-truth location of the drone. This will happen even if the drone is static and is reporting no changes in velocity. This seems to be caused by incorrect sensor readings that report obstacles as closer than they actually are, which causes the localization update to move the drone approximately the same distance as the change in obstacle distance. The system worked best when the feature observation covariance $\mathbf{R}_k$ was very high, causing the update to rely on the drone's odometry for the updated position estimate. This is disappointing, as the purpose of this project was to overcome error in the drone's odometry by using additional sensors.

Four tests were done to compare the effectiveness of the drone's odometry and the EKF-SLAM system. The drone was flown a number of times with four different system configurations, and was flown both under human control and autonomous control. Between the four system configurations, the only change was the way the $\mathbf{R}_k$ feature noise covariance matrix was scaled. With $\mathbf{R}_k$ scaled by 100, the EKF-SLAM system relies more heavily on the motion update for the new position estimate. With an $\mathbf{R}_k$ scalar of 10, the EKF-SLAM system weights feature information higher to update the drone's position estimate. All tests were done with a sensor update window length of 20, sonar filtering that keeps the maximum of every three values, and a maximum obstacle range of one meter. The maximum obstacle range of one meter was heuristically chosen based on experiments with the sonar sensors, which have greater variance past a range of one meter.

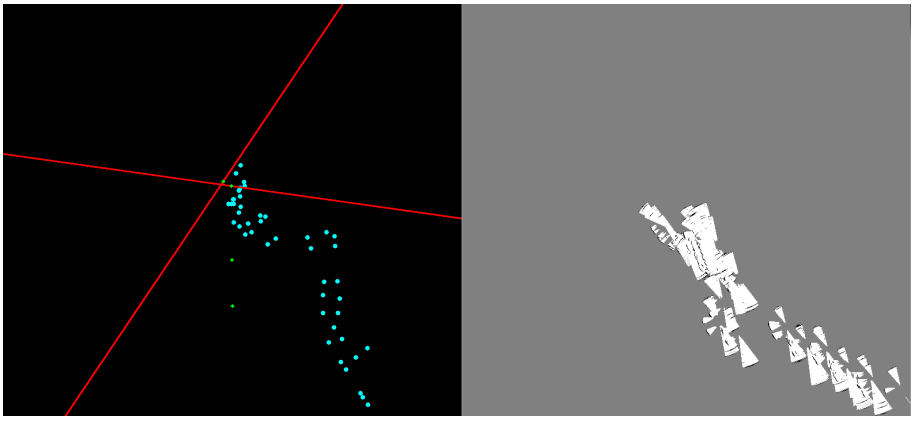

Figure 4.1: *An example of a real-world test showing drastic divergence of the drone's position.*

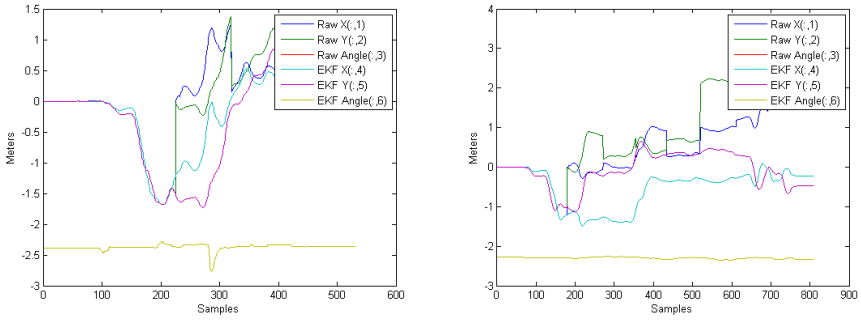

Figure 4.2: *Graphs showing the* x, y, *and* angle *of the drone's odometry and EKF-SLAM estimate. Left has an* $\mathbf{R}_k$ *scalar of 100, right has an* $\mathbf{R}_k$ *scalar of 10. It is easy to see that the* x *and* y *values differ greatly in both test cases, while the angle returned by the EKF-SLAM system stays remarkably similar to the raw odometry.*

Finally, the results of the four different configuration tests shown here:

	X (meters)	*Y (meters)*	*Angle (radians)*
Raw odometry, human controlled.	**0.384**	**0.568**	**0.180**
Raw odometry, autonomous.	**1.190**	**0.680**	**1.068**
EKF-SLAM, scalar of 100	**0.372**	**0.766**	**0.098**
EKF-SLAM, scalar of 10.	**1.1934**	**1.1882**	**0.116**

Table 4.1: *Average difference between ground-truth and estimated drone position.*

The raw odometry with human controlled motion has the lowest average error in real-world tests. The autonomous motion has higher error as the obstacle-reactive state machine

overcompensates for obstacles with large position changes compared to a human operator. When using the EKF-SLAM system with feature variance $\mathbf{R}_k$ scaled by 100, the error is similar to the error in the raw drone odometry. This is expected, as the EKF update weights the drone's odometry over the high variance in the feature observation. With $\mathbf{R}_k$ scaled by 10, the feature observation is trusted more by the EKF update and the error in the drone's estimated position increases due to inconsistent landmark observation. These results show that the sensor package was less accurate than the drone's internal odometry, and thus not able to compensate for the drone's odometry error.

The sonar package was effective at obstacle avoidance during autonomous navigation, with the drone successfully avoiding over 70% of obstacles during 30 to 60 second long tests. Small obstacles or obstacles that fit between the two front sensors would occasionally be hit while flying, and the drone would occasionally drift into obstacles while reversing or turning in tight spaces, as the turbulence caused by these motions can cause the drone to behave erratically.

4.3 Problems Encountered

There are currently some problems with the system that are interfering with successful real-world tests.

4.3.1 Sonar

The sonar sensors are accurate when perpendicular to an object. However, when mounted on the airborne drone, the sonar seem to be much more erratic. Figure 4.1 shows a graph of 100 sonar samples from each sonar taken while the drone was hovering in the air. The left and right sensors are incredibly erratic at the larger ranges, while the front sensors are more consistent. This may be due to changes in the drone's pitch, yaw, and roll, (which are unavoidable in drone movement) causing the sonar beam to change its angle relative to the detected obstacles. This will cause the returned range to be erratic. This could be due to vibration of the drone's chassis as it hovers, or possibly sonar interference from high frequency signals created by the drone or present in the environment. MaxBotix does not recommend applying filters to the sonar results, as they are already filtered in the sensor itself. However, many configurations of median and maximum filters were applied to the sonar in an effort to reduce noise. With a large filter (10 samples or greater) noise was significantly reduced, with time between effective samples increasing too high to be useful during drone movement. Also, the errors in the sonar are primarily negative, which does not work well with the EKF-SLAM system which assumes that noise is Gaussian.

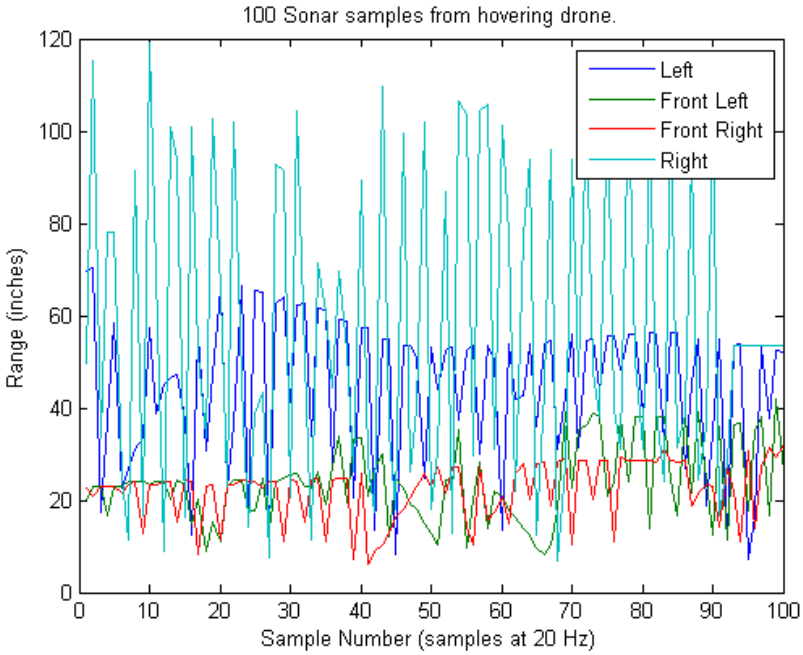

Figure 4.3: *100 sonar samples taken while the drone was hovering in a hallway. In this sample data, there seems to be periodic interference in the front left and front right sonar. The interference was not consistent between multiple tests.*

4.3.2 AR.Drone

While the AR.Drone is designed for indoor flight, the sonar sensors seem to require a greater amount of flight stability than the drone can provide. The turbulence created by the drone's quad-rotors alone cause the drone to behave erratically in small, enclosed environments such as hallways. Other airflow sources, such as heating or cooling systems, are also problematic. A few conditions were found in the AR.Drone's internal sensor fusion processes that can cause dramatic error in the system. If the drone runs into an obstacle and bounces erratically, the odometry error increases dramatically during the recovery period as the velocities are not sampled by the *Pose Estimation* module frequently enough. Additionally, low-light conditions and uniform visual environment features often present in hallways do not provide enough information for the sensor fusion built into the AR.Drone that governs its stability and feedback loops. In low-light conditions, the drone is less stable, more prone to drift, and odometry errors are higher. While tight, uniform hallways are preferable for sonar feature extraction, they are problematic for drone navigation and sensor fusion.

4.4 Future Work

Other than improving the sonar sensors used in this system, there are two additional features I would like to incorporate into this project in the future. The first is a feature that was cut from the project in its early stages, and the second is that was not implemented due to time constraints:

4.4.1 Vision-based environment categorization.

The AR.Drone sends live feeds from its front and bottom facing cameras over the wireless link to the control device. However, early in the project I switched from the AR.Drone 1.0 to the AR.Drone 2.0. One of the version 2.0 improvements was increased camera resolution. The front camera increased from 640x480 to 1280x720, and the bottom facing camera doubled in size. However, this broke video compatibility with the *libardrone* python library. The new HD video feed used a different codec, that was custom made by Parrot. I eventually decided to cancel the video portion of the project. Additionally, decoding the HD video feed with some existing tools I found took significant processing power and was very slow.

4.4.2 Map-Assisted Navigation

Once a map of the environment is built, the mapping system could be expanded to find the most efficient path back to an area it has already been. There are a variety of path-planning methods to choose from, and many could work well for the landmark system used in this project. It may be possible to use the sensed map features as nodes in a graph, and attempt to build the shortest path between the current node and desired node.

Appendix A

Obstacle Reactive Autonomous Navigation State Machine

Hover Start Clears hover counter, moves into Hover The purpose of this project is to add autonomous navigation and mapping to a micro aerial vehicle (MAV) with the aid of additional sonar sensors that are mounted to the MAV's chassis. In this paper, an Extended Kalman Filter Simultaneous Localization and Mapping (EKF-SLAM) system is implemented on a MAV with four sonar sensors. Some simulations are developed to test the various modules and processes created for the project, and the final system is tested on an AR.Drone 2.0 quadcopter with an added sonar sensor package.

Hover Stays in hover state for 20 iterations while the front sensors remain clear. If the front sensors do not remain clear, the hover counter is reset. This allows the drone to avoid mobile obstacles.

Hover Start from Reverse Clears the hover counter, moves state machine to Hover From Reverse

Hover From Reverse Stays in state for 20 iterations while the front sensors remain clear. Then, moves to Turn Left Start or Turn Right Start based on the ranges reported by the side sensors.

Forwards Drone moves forward until any of the sensors report an obstacle. When an obstacle is detected by the front sensors, the state machine is sent to the Reverse state. If

an obstacle is detected by either side sensor, the state machine moves to the appropriate Turn state.

Reverse Drone moves backward until both front sensors are clear.

Turn Right Start Capture current yaw, move to Turn Right state .

Turn Right Turn to the right at least 45 degrees, or until the left sensor is clear.

Turn Left Start Capture current yaw, move to Turn Left state .

Turn Left Turn to the left at least 45 degrees, or until the right sensor is clear.

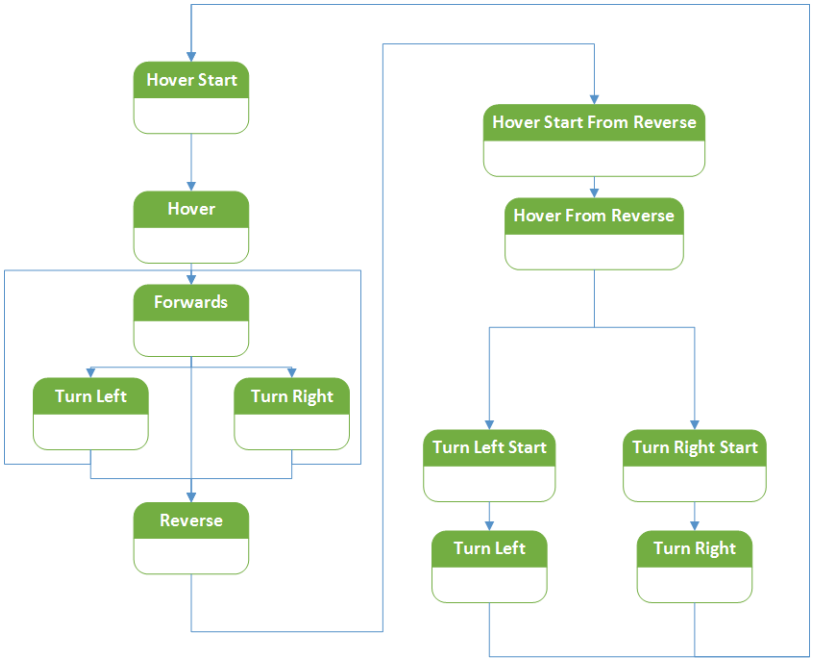

Figure A.1: *State Machine Diagram for Obstacle Reactive Autonomous Exploration*

Appendix B

AR.Drone 2.0 Motion Commands

- Forwards - move the drone forwards (w).
- Backwards - move the drone backwards (s).
- Left - slide the drone to the left without rotating (a).
- Right - slide the drone right without rotating (d).
- Turn Left - rotate the drone counter-clockwise (left arrow).
- Turn Right - rotate the drone clockwise (right arrow).
- Move Up - increase the drone's height (up arrow).
- Move Down - decrease the drone's height (down arrow).
- Take Off - tell the drone to run its take-off routine (enter)
- Land - tell the drone to run its landing routine (space bar)
- Emergency - cut power to the drone's rotors (esc).

All of these commands will move the drone at the current drone speed setting. During testing I found speed value of 0.1 useful for forward and backward movement. Before rotating the drone the speed is changed to 0.5 for improved turning accuracy, as lower speeds such as 0.1 caused the drone to drift erratically while turning. When the turn is complete, drone speed is set back to 0.1.

Bibliography

[1] C. Bills, J. Chen, and A. Saxena, "Autonomous mav flight in indoor environments using single image perspective cues," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 5776–5783.

[2] S. Shen, N. Michael, and V. Kumar, "Autonomous indoor 3d exploration with a micro-aerial vehicle," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 9–15.

[3] T. KrajnÃk, V. VonÃ¡sek, D. Fiser, and J. Faigl, in *Eurobot Conference*, ser. Communications in Computer and Information Science, D. ObdrzÃ¡lek and A. Gottscheber, Eds. Springer, pp. 172–186.

[4] M. Drumheller, "Mobile robot localization using sonar," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, pp. 325–332, 1987.

[5] V. Varveropoulos, "Robot localization and map construction using sonar data," April 2001.

[6] J. Muller and W. Burgard, "Efficient probabilistic localization for autonomous indoor airships using sonar, air flow, and imu sensors."

[7] J. D. Tardos, J. Neira, P. M. Newman, and J. J. Leonard, "Robust mapping and localization in indoor environments using sonar data," *Int. J. Robotics Research*, vol. 21, pp. 311–330, 2002.

[8] MaxBotix, "Lv-maxsonar-ez0 high performmance sonar range finder mb1000."